

Standard Code Library

FLself

SCUT

September 1, 2022

Contents

一切的开始	2
一些宏定义	2
数据结构	3
ST 表	3
Fenwick Tree(树状数组)	4
线段树	4
主席树	5
数学	5
数论	5
数论整数	5
拉格朗日差值法	6
欧几里得	6
中国剩余定理	7
各种筛	7
多项式	10
组合数学	14
卡特兰数	14
斯特林数	14
分拆数	15
容斥原理	15
线性代数	16
矩阵	16
线性基	17
博弈论	18
图论	18
LCA	18
连通性	19
tarjan	19
树	19
虚树	19
网络流	21
Dinic	21
计算几何	22
二维几何: 点与向量	22
字符串	23
KMP	23
manacher	23
后缀自动机	23
回文自动机	25
杂项	26
STL	26

一切的开始

一些宏定义

- 需要 C++11

```
1 // #define DEBUG
2 // #define InTerminal
3 // #define std_cpp17
4 #include<bits/stdc++.h>
5 #define int long long
6 #define PII std::pair<int, int>
7 #define VI std::vector<int>
8 #define VPII std::vector<std::pair<int, int> >
9 #define VVI std::vector<std::vector<int> >
10 #define ALL(a) (a).begin(), (a).end()
11 #define SIZ(a) ((int)(a).size())
12 #define FOR(i, l, r) for (int i = (l); i <= (r); ++i)
13 #define REP(i, r, l) for (int i = (r); i >= (l); --i)
14 #define lowbit(x) ((x) & -(x))
15 #define lbpos(x) (__builtin_ctz(x))
16 #define hbpos(x) (31 - __builtin_clz(x))
17
18 template<typename S, typename T> std::istream &operator>>(std::istream &is, std::pair<S, T> &pp) { is >> pp.first >>
    ⇨ pp.second; return is; }
19 template<typename S, typename T> std::ostream &operator<<(std::ostream &os, std::pair<S, T> pp) { os << "(" <<
    ⇨ pp.first << ", " << pp.second << ")"; return os; }
20 template<typename S, std::size_t _siz> std::istream &operator>>(std::istream &is, std::array<S, _siz> &arr) { for
    ⇨ (auto &x: arr) is >> x; return is; }
21 template<typename S, std::size_t _siz> std::ostream &operator<<(std::ostream &os, std::array<S, _siz> arr) { os <<
    ⇨ "("; for (auto x: arr) os << x << ", "; os << ")"; return os; }
22 template<typename T> std::istream &operator>>(std::istream &is, std::vector<T> &vec) { for (auto &x: vec) is >> x;
    ⇨ return is; }
23 template<typename T> std::ostream &operator<<(std::ostream &os, const std::vector<T> &vec) { os << '{'; for (auto
    ⇨ &x: vec) os << x << ", "; return os << "}"; }
24 #ifndef std_cpp17
25 template<class Tuple, std::size_t... Is> void print_tuple_impl(std::ostream &os, const Tuple &t,
    ⇨ std::index_sequence<Is...>) { ((os << (Is == 0? "" : ", ") << std::get<Is>(t), ...); }
26 template<class... Args> std::ostream &operator<<(std::ostream &os, const std::tuple<Args...> &t) { os << "(";
    ⇨ print_tuple_impl(os, t, std::index_sequence_for<Args...>{}); return os << ")"; }
27 #endif
28 #ifndef DEBUG
29 #ifndef InTerminal
30 #define dbg(x...) do { std::cerr << "\033[32;1m" << #x << " -> "; err(x); } while (0)
31 void err() { std::cerr << "\033[39;0m" << std::endl; }
32 #else
33 #define dbg(x...) do { std::cerr << #x << " -> "; err(x); } while (0)
34 void err() { std::cerr << std::endl; }
35 #endif
36 template<typename T, typename... A>
37 void err(T a, A... x) { std::cerr << a << ' '; err(x...); }
38 #else
39 #define dbg(...)
40 #endif
41
42 using namespace std;
43 const int maxn = 2e5 + 3;
44 const int INF = 0x3f3f3f3f3f3f3f3f;
45 const int mod = 998244353;
46 mt19937 RD(time(0));
47
48
49
50 void solv() {
51
52
53     return ;
54 }
55
56 signed main() {
57     // freopen("./data.in", "r", stdin);
```

```

58     std::ios::sync_with_stdio(false), std::cin.tie(0), std::cout.tie(0);
59     int beg__TT = clock();
60
61     signed _ttt;
62     cin >> _ttt;
63
64     while(_ttt--)
65         solv();
66
67     #ifdef DEBUG
68     std::cerr << "use : " << (clock() - beg__TT) << "ms\n";
69     #endif
70     return 0;
71 }

```

数据结构

ST 表

- 一维

```

1  class Sparcetable {
2      vector<vector<int> > st;
3      int siz;
4      bool MX_flg = 0;
5      inline int renew(int x, int y) {
6          if (MX_flg) return max(x, y);
7          return min(x, y);
8      }
9  public:
10     // 注意 bhpos(0) 返回-1
11     bool (*comp)(int, int);
12     Sparcetable():siz(maxn) {st.resize(hbpos(maxn - 1) + 1, std::vector<int> (maxn));}
13     Sparcetable(const std::vector<int>& a, bool _MX_flg = 1): siz(a.size()), MX_flg(_MX_flg) {
14         int n = a.size();
15         st.resize(hbpos(n) + 1, vector<int> (n + 1));
16         for (int i = 1; i <= n; ++i) st[0][i] = a[i];
17         for (int i = 1; i <= hbpos(siz); ++i) {
18             for (int j = 1; j + (1 << i) <= siz + 1; ++j) {
19                 st[i][j] = renew(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
20             }
21         }
22     }
23     int query(int l, int r) {
24         int len = hbpos(r - l + 1);
25         return renew(st[len][l], st[len][r - (1 << len) + 1]);
26     }
27 };

```

- 二维

```

1  int f[10][10][maxn][maxn];
2  #define _highbit(x) (31 - __builtin_clz(x))
3  inline int calc(int x, int y, int xx, int yy, int p, int q) {
4      return max(
5          max(f[p][q][x][y], f[p][q][xx - (1 << p) + 1][yy - (1 << q) + 1]),
6          max(f[p][q][xx - (1 << p) + 1][y], f[p][q][x][yy - (1 << q) + 1])
7      );
8  }
9  void init() {
10     for (int x = 0; x <= _highbit(n); ++x)
11     for (int y = 0; y <= _highbit(m); ++y)
12     for (int i = 0; i <= n - (1 << x); ++i)
13     for (int j = 0; j <= m - (1 << y); ++j) {
14         if (!x && !y) { f[x][y][i][j] = a[i][j]; continue; }
15         f[x][y][i][j] = calc(
16             i, j,
17             i + (1 << x) - 1, j + (1 << y) - 1,
18             max(x - 1, 0), max(y - 1, 0)
19         );

```

```

20     }
21 }
22 inline int get_max(int x, int y, int xx, int yy) {
23     return calc(x, y, xx, yy, _highbit(xx - x + 1), _highbit(yy - y + 1));
24 }

```

Fenwick Tree(树状数组)

- 一维

```

1  template<typename T>
2  class FenwickT {
3      int n;
4      vector<T> tr;
5  public:
6      FenwickT(int siz): tr(siz), n(siz) {}
7      FenwickT(int siz, T ini): tr(siz, ini), n(siz) {}
8      void add(int p, T x) {
9          for (int i = p; i < n; i += i & (-i)) tr[i] += x;
10     }
11     T query(int p) {
12         T ret = T(0);
13         for (int i = p; i > 0; i -= i & (-i)) ret += tr[i];
14         return ret;
15     }
16     T range_sum(int l, int r) {
17         return (query(r) - query(l - 1));
18     }
19 };

```

线段树

```

1  class SegT{
2      struct Pt{
3          int val, ls, rs, sl;
4          Pt():val(0), ls(0), rs(0), sl(0) {}
5          Pt(int v, int l, int r, int s): val(v), ls(l), rs(r), sl(s) {}
6          ~Pt() {}
7      };
8      int renew(int x, int y) {
9          return x + y;
10     }
11     int renewsl(int x, int y) {
12         return x + y;
13     }
14     int renewqj(int x, int y, int l, int r) {
15         return x + y * (r - l + 1);
16     }
17  public:
18      std::vector<Pt> tr;
19      int cnt = 0; // 结点数
20      SegT() {cnt = 1; tr.emplace_back();} // 建一棵空树
21
22      // 修改
23      void add(int l, int r, int p, int ql, int qr, int x) {
24          if (ql <= l && r <= qr) {
25              tr[p].val = renewqj(tr[p].val, x, l, r);
26              tr[p].sl = renewsl(tr[p].sl, x);
27              return;
28          }
29          int mid = l + (r - l) / 2;
30          if (tr[p].sl) {
31              if (!tr[p].ls) tr[p].ls = cnt++;
32              if (!tr[p].rs) tr[p].rs = cnt++;
33              if (cnt >= tr.size()) tr.resize(cnt + 1);
34              int ls = tr[p].ls, rs = tr[p].rs;
35              tr[ls].val = renewqj(tr[ls].val, tr[p].sl, l, mid), tr[rs].val = renewqj(tr[rs].val, tr[p].sl, mid + 1, r);
36              tr[ls].sl = renewsl(tr[ls].sl, tr[p].sl), tr[rs].sl = renewsl(tr[rs].sl, tr[p].sl);
37              tr[p].sl = 0;
38          }

```

```

39     if (ql <= mid) {
40         if (!tr[p].ls) tr[p].ls = cnt++;
41         if (cnt >= tr.size()) tr.resize(cnt + 1);
42         add(l, mid, tr[p].ls, ql, qr, x);
43     }
44     if (qr > mid) {
45         if (!tr[p].rs) tr[p].rs = cnt++;
46         if (cnt >= tr.size()) tr.resize(cnt + 1);
47         add(mid + 1, r, tr[p].rs, ql, qr, x);
48     }
49     tr[p].val = renew(tr[p].ls? tr[tr[p].ls].val: 0, tr[p].rs? tr[tr[p].rs].val: 0);
50 };
51
52 // 查询
53 int query(int l, int r, int p, int ql, int qr) {
54     if (ql <= l && r <= qr) return tr[p].val;
55     int mid = l + (r - l) / 2;
56     if (tr[p].sl) {
57         if (!tr[p].ls) tr[p].ls = cnt++;
58         if (!tr[p].rs) tr[p].rs = cnt++;
59         if (cnt >= tr.size()) tr.resize(cnt + 1);
60         int ls = tr[p].ls, rs = tr[p].rs;
61         tr[ls].val = renewqj(tr[ls].val, tr[p].sl, l, mid), tr[rs].val = renewqj(tr[rs].val, tr[p].sl, mid + 1, r);
62         tr[ls].sl = renewsl(tr[ls].sl, tr[p].sl), tr[rs].sl = renewsl(tr[rs].sl, tr[p].sl);
63         tr[p].sl = 0;
64     }
65     int ret = 0;
66     if (ql <= mid) {
67         if (!tr[p].ls) tr[p].ls = cnt++;
68         if (cnt >= tr.size()) tr.resize(cnt + 1);
69         ret = renew(ret, query(l, mid, tr[p].ls, ql, qr));
70     }
71     if (qr > mid) {
72         if (!tr[p].rs) tr[p].rs = cnt++;
73         if (cnt >= tr.size()) tr.resize(cnt + 1);
74         ret = renew(ret, query(mid + 1, r, tr[p].rs, ql, qr));
75     }
76     tr[p].val = renew(tr[p].ls? tr[tr[p].ls].val: 0, tr[p].rs? tr[tr[p].rs].val: 0);
77     return ret;
78 };
79 };

```

主席树

可持久化线段树, 感觉主要是利用单点 \log 次的性质以及离散化的思路. 比如区间第 k 小就是构造 n 个线段树 (每多加一个实际上只多加了 \log 个点), 然后再用离散化的点 (实际上就是其排名值) 找到前缀和之差为 k (此时为第 k 小, 可以这样想: 第 r 棵树的 r 个点中包含了第 $l-1$ 棵树的 $l-1$ 个点, 减掉那些点, 剩下的就是区间 $[l, r]$ 的点, 然后找第 k 个) 的点. 用离线不那么方便. 代码暂时先留空了...

数学

数论

数论整数

```

1  constexpr int P = 998244353;
2  // assume -P <= x < 2P
3  int norm(int x) {
4      // x %= P;
5      if (x < 0) { x += P; }
6      if (x >= P) { x -= P; }
7      return x;
8  }
9  template<typename E>
10 E power(E n, int k) {
11     E ret = E(1);
12     while (k) {
13         if (k & 1) ret *= n;

```

```

14         n *= n;
15         k >>= 1;
16     } return ret;
17 }
18 struct Z {
19     int x;
20     Z(int x = 0) : x(norm(x)) {}
21     int val() const { return x; }
22     Z operator-() const { return Z(norm(P - x)); }
23     Z inv() const { assert(x != 0); return power(*this, P - 2); }
24     Z &operator*=(const Z &rhs) { x = (long long)(x) * rhs.x % P; return *this; }
25     Z &operator+=(const Z &rhs) { x = norm(x + rhs.x); return *this; }
26     Z &operator-=(const Z &rhs) { x = norm(x - rhs.x); return *this; }
27     Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
28     friend Z operator*(const Z &lhs, const Z &rhs) { Z res = lhs; res *= rhs; return res; }
29     friend Z operator+(const Z &lhs, const Z &rhs) { Z res = lhs; res += rhs; return res; }
30     friend Z operator-(const Z &lhs, const Z &rhs) { Z res = lhs; res -= rhs; return res; }
31     friend Z operator/(const Z &lhs, const Z &rhs) { Z res = lhs; res /= rhs; return res; }
32     friend std::istream &operator>>(std::istream &is, Z &a) { long long v; is >> v; a = Z(v); return is; }
33     friend std::ostream &operator<<(std::ostream &os, const Z &a) { return os << a.val(); }
34 };

```

拉格朗日差值法

欧几里得

- 扩展欧几里得

```

1 int exgcd(int a, int b, int& x, int& y) {
2     if (a == 0) {
3         x = 0, y = 1;
4         return b;
5     }
6     int ret = exgcd(b % a, a, x, y), xx = x;
7     x = y - b / a * x;
8     y = xx;
9     return ret;
10 }

```

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ 或 $b \geq c$ 时, $f(a, b, c, n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$; 否则 $f(a, b, c, n) = nm - f(c, c-b-1, a, m-1)$ 。
- $g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ 或 $b \geq c$ 时, $g(a, b, c, n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$; 否则 $g(a, b, c, n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$ 。
- $h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \geq c$ 或 $b \geq c$ 时, $h(a, b, c, n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2 (n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$; 否则 $h(a, b, c, n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a, b, c, n)$ 。

```

1 template<typename T>
2 T lintp(const vector<int>& x, const vector<T>& y, int k) {
3     T ans = 0;
4     for (int i = 0; i < x.size(); ++i) {
5         if (k == x[i])
6             return y[i];
7         T u = 1, v = 1;
8         for (int j = 0; j < x.size(); ++j) {
9             if (i == j) continue;
10            u *= (k - x[j]);
11            v *= (x[i] - x[j]);
12        }
13        ans += y[i] * u / v;
14    }
15    return ans;
16 }

```

中国剩余定理

```

1  int CRT(std::vector<int> a, std::vector<int> r) {
2      long long n = 1, ans = 0, k = a.size();
3      for (int i = 0; i < k; ++i) n = n * r[i];
4      for (int i = 0; i < k; ++i) {
5          long long m = n / r[i], b, y;
6          exgcd(m, r[i], b, y);
7          ans = (ans + a[i] * m % n * b % n + n) % n;
8      }
9      return ans;
10 }

```

各种筛

筛法	场景	效率
Min-25 筛	$f(p)$ 是一个关于 p 的多项式, $f(p^c)$ 能快速求	$O(\frac{n^{\frac{3}{4}}}{\log n})$
PN 筛	找一个好求前缀和的积性函数 $g()$ 在 p 处 $f(p) = g(p)$	$O(\sqrt{n})$
杜教筛	找一个 $g()$ 使得 $f * g()$ 好求前缀和	$O(n^{\frac{2}{3}})$

Min-25 筛

$$\begin{aligned}
 F_k(n) &= \sum_{i=2}^n [p_k \leq \text{lpf}(i)] f(i) \\
 &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + \sum_{\substack{k \leq i \\ p_i \leq n}} f(p_i) \\
 &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \\
 &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^{c+1} \leq n}} (f(p_i^c) F_{i+1}(n/p_i^c) + f(p_i^{c+1})) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \\
 G_k(n) &= G_{k-1}(n) - [p_k^2 \leq n] g(p_k) (G_{k-1}(n/p_k) - G_{k-1}(p_{k-1}))
 \end{aligned}$$

```

1  namespace Min_25{
2      bool INI = 0;
3      int M = 0, sqn = 0;
4      std::vector<int> primes;
5      Z (*fp)(int, int); // f(p^c)
6      Z fp_single(int p, int c) {
7          return power(Z(p), c);
8      }
9      Z sgp(int n, int c) {
10         if (c == 0) return Z(n);
11         if (c == 1) return Z(n+1) * n / 2;
12         if (c == 2) return Z(2*n+1) * (n+1) * n / 6;
13         // if (c == 3) return Z(n) * n * (n+1) * (n+1) / 4;
14         // if (c == 4) return Z(n) * (n+1) * (2*n+1) * (3*n*n%P + 3*n - 1) / 30;
15     }
16     struct Gpoly{
17         int M, sqn, a;
18         std::vector<Z> F_prime1, F_prime2;
19         Gpoly(int scale, int a, int c): M(scale), a(a) {
20             sqn = sqrt(M) + 3;
21             F_prime1.resize(sqn), F_prime2.resize(sqn);
22         }
23         int st = M, ed = 1;
24         for (int i = 1; st >= ed; st = min(M / ++i, st - 1)) {
25             (*this)[st] = sgp(st, c) - 1;
26         }
27         for (int j = 1; j < primes.size(); ++j) {

```



```

28         st = M, ed = primes[j] * primes[j];
29         for (int i = 1; st >= ed; st = min(M / ++i, st - 1)) {
30             (*this)[st] -= fp_single(primes[j], c) * ((*this)[st/primes[j]] - (*this)[primes[j-1]]);
31         }
32     }
33 }
34 void seta(int aa) {a = aa;}
35 Z & operator [] (int idx) {
36     if (idx < sqn) return F_prime1[idx];
37     return F_prime2[M/idx];
38 }
39 };
40
41 std::vector<Gpoly> Gp;
42 Z G(int x){
43     Z res = 0;
44     for (auto &g: Gp) res += g[x] * g.a;
45     return res;
46 }
47 void init(int scale) {
48     if (M < scale) INI = 0;
49     M = scale, sqn = sqrt(M) + 5;
50
51     if (!INI) {
52         INI = 1;
53         std::vector<signed> vis(sqn);
54         primes.resize(1, 1);
55         for (int i = 2; i < sqn; ++i) {
56             if (!vis[i]) primes.push_back(i);
57             for (int j = 1; j < primes.size(); ++j) {
58                 auto prm = primes[j];
59                 if (prm * i >= sqn) break;
60                 vis[i * prm] = 1;
61                 if (i % prm == 0) break;
62             }
63         }
64     }
65
66     Gp.clear();
67     Gp.emplace_back(scale, N, 0);
68     // Gp.emplace_back(scale, a, c); // a*p^c
69 }
70
71 Z seive(int n, int k) {
72     Z res = G(n) - G(primes[k-1]);
73     for (int i = k; i < primes.size() && primes[i] * primes[i] <= n; ++i) {
74         Z fpj = fp(primes[i], 1), fpj1;
75         for (int j = 1, pc = primes[i]; pc * primes[i] <= n; ++j, pc *= primes[i], fpj = fpj1) {
76             fpj1 = fp(primes[i], j + 1);
77             res += fpj * seive(n/pc, i + 1) + fpj1;
78         }
79     }
80     return res;
81 }
82 Z S_f(int n) {
83     return seive(n, 1) + 1;
84 }
85 void seta(const vector<int>& a) {for (int i = 0; i < Gp.size() && i < a.size(); ++i) Gp[i].seta(a[i]);}
86 }

```

PN 筛 (Powerful Number)

```

1 // f = (g*h); f(p) = g(p) + h(p) [with f(p) = g(p)] -> h[p] = 0.
2 namespace PowerfulNumber{
3     bool INI = 0;
4     std::vector<int> primes;
5     std::vector<std::vector<Z>> h;
6     int M;
7     Z (*fp)(int, int); // f(p^c)
8     Z (*gp)(int, int); // g(p^c)
9     Z (*S_g)(int); // preffix sum of g()

```

```

10
11 // if lack of a formula of h(i, j), you need to set f() and g() beforehand
12 void init(int scale) {
13     if (M < scale) INI = 0;
14     M = scale;
15     int n = std::sqrt(M)+10;
16     if (!INI) {
17         INI = 1;
18         primes.resize(0);
19         std::vector<signed> vis(n);
20         for (int i = 2; i < n; ++i) {
21             if (!vis[i]) primes.push_back(i);
22             for (auto prm: primes) {
23                 if (prm * i >= n) break;
24                 vis[i * prm] = 1;
25                 if (i % prm == 0) break;
26             }
27         }
28         h.resize(primes.size(), std::vector<Z>((int)(log2(M))+1));
29     }
30
31     // get the function h() (with f() and g() set or with formula of h())
32     for (int i = 0; i < h.size(); ++i) {
33         int pp = primes[i] * primes[i];
34         if (pp > M) break;
35         h[i][0] = 1;
36         for (int j = 2; pp <= M && j < h[i].size(); ++j, pp *= primes[i]) {
37             Z sgh = 0;
38             for (int k = 1, xp = primes[i]; k <= j; ++k, xp *= primes[i]) {
39                 sgh += gp(primes[i], xp) * h[i][j-k];
40             }
41             h[i][j] = fp(pp, j) - sgh;
42             // h[i][j] = Z(j-1) * (pp * primes[i] % P - pp % P);
43             // h[i][j] = Z(-pp) / (j * (j-1));
44             // [a formula of h(i, j)], better faster than log, this example can be optimized to O(1).
45         }
46     }
47 }
48
49 // assistance func to get the sum
50 Z PN_sieve(int n, int flr, Z hd) {
51     Z res = S_g(n)*hd;
52     for(int i = flr+1; i < primes.size(); ++i) {
53         int prm=primes[i], k=1;
54         int val=n/prm, pk=prm;
55
56         if(val < prm) break;
57         while(val >= prm) {
58             val /= prm;
59             pk *= prm;
60             ++k;
61
62             res += PN_sieve(val, i, hd*h[i][k]);
63         }
64     }
65     return res;
66 }
67 // func to get the sum
68 Z getsum(int n) {
69     return PN_sieve(n, -1, 1);
70 }
71 }

```

杜教筛

```

1 // g(1)S(n) = \sum_{i=1}^n (f*g)(i) - \sum_{i=2}^n g(i)S(n/i)
2 namespace dujiaoshai{
3     vector<Z> smalls;
4     map<int, Z> bigS;
5
6     void Set_smalls(int);

```

```

7   Z (*S_fg)(int);
8   Z (*S_g)(int);
9   Z getS(int n) {
10      if (n < smallS.size()) return smallS[n];
11      else if (bigS.count(n)) return bigS[n];
12      Z res = S_fg(n);
13      for (int l = 2, r; l <= n; l = r + 1) {
14          r = n / (n / l);
15          res -= (S_g(r) - S_g(l-1)) * getS(n/l);
16      }
17      // res /= S_g(1);
18      return (bigS[n] = res);
19  }
20 }
21
22 void dujiaoshai::Set_smallS(int siz) {
23     int n = pow(siz, 0.67);
24     smallS.assign(n, 0);
25     smallS[1] = 1;
26     vector<signed> vis(n, primes);
27     for (int i = 2; i < n; ++i) {
28         if (!vis[i]) {
29             primes.push_back(i);
30             smallS[i] = i - 1;
31         }
32         for (auto &prm: primes) {
33             if (i * prm >= n) break;
34             vis[i * prm] = 1;
35             if (i % prm == 0) {
36                 smallS[i * prm] = smallS[i] * prm;
37                 break;
38             }
39             smallS[i * prm] = smallS[i] * (prm - 1);
40         }
41         smallS[i] = smallS[i-1] + smallS[i]*i;
42     }
43 }

```

多项式

• Poly with NTT

```

1   std::vector<int> rev;
2   std::vector<Z> roots{0, 1};
3   void dft(std::vector<Z> &a) {
4       int n = a.size();
5
6       if ((int)(rev.size()) != n) {
7           int k = __builtin_ctz(n) - 1;
8           rev.resize(n);
9           for (int i = 0; i < n; i++) {
10              rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
11          }
12      }
13
14      for (int i = 0; i < n; i++) {
15          if (rev[i] < i) {
16              std::swap(a[i], a[rev[i]]);
17          }
18      }
19      if ((int)(roots.size()) < n) {
20          int k = __builtin_ctz(roots.size());
21          roots.resize(n);
22          while ((1 << k) < n) {
23              Z e = power(Z(3), (P - 1) >> (k + 1));
24              for (int i = 1 << (k - 1); i < (1 << k); i++) {
25                  roots[2 * i] = roots[i];
26                  roots[2 * i + 1] = roots[i] * e;
27              }
28              k++;
29          }

```

```

30     }
31     for (int k = 1; k < n; k *= 2) {
32         for (int i = 0; i < n; i += 2 * k) {
33             for (int j = 0; j < k; j++) {
34                 Z u = a[i + j];
35                 Z v = a[i + j + k] * roots[k + j];
36                 a[i + j] = u + v;
37                 a[i + j + k] = u - v;
38             }
39         }
40     }
41 }
42 void idft(std::vector<Z> &a) {
43     int n = a.size();
44     std::reverse(a.begin() + 1, a.end());
45     dft(a);
46     Z inv = (1 - P) / n;
47     for (int i = 0; i < n; i++) {
48         a[i] *= inv;
49     }
50 }
51 struct Poly {
52     std::vector<Z> a;
53     Poly() {}
54     Poly(const std::vector<Z> &a) : a(a) {}
55     Poly(const std::initializer_list<Z> &a) : a(a) {}
56     int size() const {
57         return a.size();
58     }
59     void resize(int n) {
60         a.resize(n);
61     }
62     Z operator[](int idx) const {
63         if (idx < size()) {
64             return a[idx];
65         } else {
66             return 0;
67         }
68     }
69     Z &operator[](int idx) {
70         return a[idx];
71     }
72     Poly mulxk(int k) const {
73         auto b = a;
74         b.insert(b.begin(), k, 0);
75         return Poly(b);
76     }
77     Poly modxk(int k) const {
78         k = std::min(k, size());
79         return Poly(std::vector<Z>(a.begin(), a.begin() + k));
80     }
81     Poly divxk(int k) const {
82         if (size() <= k) {
83             return Poly();
84         }
85         return Poly(std::vector<Z>(a.begin() + k, a.end()));
86     }
87     friend Poly operator+(const Poly &a, const Poly &b) {
88         std::vector<Z> res(std::max(a.size(), b.size()));
89         for (int i = 0; i < (int)res.size(); i++) {
90             res[i] = a[i] + b[i];
91         }
92         return Poly(res);
93     }
94     friend Poly operator-(const Poly &a, const Poly &b) {
95         std::vector<Z> res(std::max(a.size(), b.size()));
96         for (int i = 0; i < (int)res.size(); i++) {
97             res[i] = a[i] - b[i];
98         }
99         return Poly(res);
100 }

```

```

101 friend Poly operator*(Poly a, Poly b) {
102     if (a.size() == 0 || b.size() == 0) {
103         return Poly();
104     }
105     int sz = 1, tot = a.size() + b.size() - 1;
106     while (sz < tot) {
107         sz *= 2;
108     }
109     a.a.resize(sz);
110     b.a.resize(sz);
111     dft(a.a);
112     dft(b.a);
113     for (int i = 0; i < sz; ++i) {
114         a.a[i] = a[i] * b[i];
115     }
116     idft(a.a);
117     a.resize(tot);
118     return a;
119 }
120 friend Poly operator*(Z a, Poly b) {
121     for (int i = 0; i < (int)(b.size()); i++) {
122         b[i] *= a;
123     }
124     return b;
125 }
126 friend Poly operator*(Poly a, Z b) {
127     for (int i = 0; i < (int)(a.size()); i++) {
128         a[i] *= b;
129     }
130     return a;
131 }
132 Poly &operator+=(Poly b) {
133     return (*this) = (*this) + b;
134 }
135 Poly &operator-=(Poly b) {
136     return (*this) = (*this) - b;
137 }
138 Poly &operator*=(Poly b) {
139     return (*this) = (*this) * b;
140 }
141 Poly deriv() const {
142     if (a.empty()) {
143         return Poly();
144     }
145     std::vector<Z> res(size() - 1);
146     for (int i = 0; i < size() - 1; ++i) {
147         res[i] = (i + 1) * a[i + 1];
148     }
149     return Poly(res);
150 }
151 Poly integr() const {
152     std::vector<Z> res(size() + 1);
153     for (int i = 0; i < size(); ++i) {
154         res[i + 1] = a[i] / (i + 1);
155     }
156     return Poly(res);
157 }
158 Poly inv(int m) const {
159     Poly x{a[0].inv()};
160     int k = 1;
161     while (k < m) {
162         k *= 2;
163         x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
164     }
165     return x.modxk(m);
166 }
167 Poly log(int m) const {
168     return (deriv() * inv(m)).integr().modxk(m);
169 }
170 Poly exp(int m) const {
171     Poly x{1};

```

```

172     int k = 1;
173     while (k < m) {
174         k *= 2;
175         x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
176     }
177     return x.modxk(m);
178 }
179 Poly pow(int k, int m) const {
180     int i = 0;
181     while (i < size() && a[i].val() == 0) {
182         i++;
183     }
184     if (i == size() || 1LL * i * k >= m) {
185         return Poly(std::vector<Z>(m));
186     }
187     Z v = a[i];
188     auto f = divxk(i) * v.inv();
189     return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
190 }
191 Poly sqrt(int m) const {
192     Poly x{1};
193     int k = 1;
194     while (k < m) {
195         k *= 2;
196         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
197     }
198     return x.modxk(m);
199 }
200 Poly mult(Poly b) const {
201     if (b.size() == 0) {
202         return Poly();
203     }
204     int n = b.size();
205     std::reverse(b.a.begin(), b.a.end());
206     return ((*this) * b).divxk(n - 1);
207 }
208 std::vector<Z> eval(std::vector<Z> x) const {
209     if (size() == 0) {
210         return std::vector<Z>(x.size(), 0);
211     }
212     const int n = std::max((int)(x.size()), size());
213     std::vector<Poly> q(4 * n);
214     std::vector<Z> ans(x.size());
215     x.resize(n);
216     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
217         if (r - l == 1) {
218             q[p] = Poly{1, -x[l]};
219         } else {
220             int m = (l + r) / 2;
221             build(2 * p, l, m);
222             build(2 * p + 1, m, r);
223             q[p] = q[2 * p] * q[2 * p + 1];
224         }
225     };
226     build(1, 0, n);
227     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
228         if (r - l == 1) {
229             if (l < (int)(ans.size())) {
230                 ans[l] = num[0];
231             }
232         } else {
233             int m = (l + r) / 2;
234             work(2 * p, l, m, num.mult(q[2 * p + 1]).modxk(m - l));
235             work(2 * p + 1, m, r, num.mult(q[2 * p]).modxk(r - m));
236         }
237     };
238     work(1, 0, n, mult(q[1].inv(n)));
239     return ans;
240 }
241 };

```

• FFT

```

1  namespace FFT { // n_ 是初始的数组长度, 不一定为 2 的幂次; n 是 init 之后的长度, 保证为 2 的幂次长度
2      const double PI = acos(-1);
3      int rev[1 << 20];
4      int init(int n_) {
5          int step = 0, n = 1;
6          for (; n < n_; n <= 1) ++step;
7          for (int i = 1; i < n; ++i) {
8              rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
9          }
10         return n;
11     }
12     void FFT(complex<double> a[], int n, int f) {
13         for (int i = 0; i < n; ++i) {
14             if (i < rev[i]) std::swap(a[i], a[rev[i]]);
15         }
16         for (int h = 2; h <= n; h <= 1) {
17             complex<double> wn(cos(f * 2 * PI / h), sin(f * 2 * PI / h));
18             for (int i = 0; i < n; i += h) {
19                 complex<double> w(1, 0), u;
20                 for (int j = i, k = h >> 1; j < i + k; ++j) {
21                     u = a[j + k] * w;
22                     a[j + k] = a[j] - u;
23                     a[j] = a[j] + u;
24                     w = w * wn;
25                 }
26             }
27         }
28         if (f == -1) {
29             for (int i = 0; i < n; ++i) {
30                 a[i] = {a[i].real() / n, 0};
31             }
32         }
33     }
34     void conv(complex<double> a[], complex<double> b[], int n_) { // n_ does not represent the pow of 2.
35         int n = init(n_);
36         FFT(a, n, 1);
37         FFT(b, n, 1);
38         for (int i = 0; i < n; ++i) a[i] *= b[i];
39         FFT(a, n, -1);
40     }
41 }

```

组合数学

卡特兰数

可以将问题划分为两个子问题的问题, 满足递推式:

$$H_n = \begin{cases} \sum_{i=1}^n H_{i-1}H_{n-i} & n \geq 2, n \in \mathbb{N}_+ \\ 1 & n = 0, 1 \end{cases}$$

递推式: $H_n = \frac{H_{n-1}(4n-2)}{n+1}$ 通项式: $H_n = \binom{2n}{n} - \binom{2n}{n-1}$

斯特林数

将 n 个不相同的元素划分为 k 个互不区分的集合, $\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$, 也记作 $S(n, k)$.

递推式: $S(n, k) = S(n-1, k) + k \times S(n-1, k)$ 通项公式: $\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{i=0}^m \frac{(-1)^{m-i} i^n}{i!(m-i)!}$

第二类斯特林数

```

1 struct Stirling2 {
2     std::vector<std::vector<Z> > S;
3     Stirling2 (int n) {
4         S.resize(n+1, vector<Z>(n+1));
5         S[0][0] = 1;
6         for (int i = 1; i <= n; ++i) {
7             for (int j = 1; j <= n; ++j) {
8                 S[i][j] = S[i-1][j-1] + S[i-1][j] * j;
9             }
10        }
11    }
12    std::vector<Z> &operator [] (int idx) {return S[idx];}
13 };
14
15
16 struct Stirling2_online {
17     std::vector<Z> S;
18     int n;
19     // S[n][k] = \sum \frac{i^n (-1)^{m-i}}{(m-i)!} i! (m-i)!
20     // that is convolution of f(x) = x^n / (x!) and g(x) = (-1)^x / (x!)
21     Stirling2_online(int _n): n(_n) {
22         vector<Z> f(n+1), g(n+1), ifact(n+1);
23         g[0] = 1;
24
25         Z fact = 1;
26         for (int i = 1; i <= n; ++i) fact *= i;
27         ifact[n] = fact.inv();
28         for (int i = n-1; i >= 1; --i) {
29             ifact[i] = ifact[i+1] * (i+1);
30         }
31
32         for (int i = 1; i <= n; ++i) {
33             f[i] = power(Z(i), n) * ifact[i];
34             g[i] = ifact[i] * (i % 2 ? -1 : 1);
35         }
36
37         S = (Poly(f) * Poly(g)).modxx(n+1).a;
38     }
39     Z operator [] (int idx) {return S[idx];}
40 };

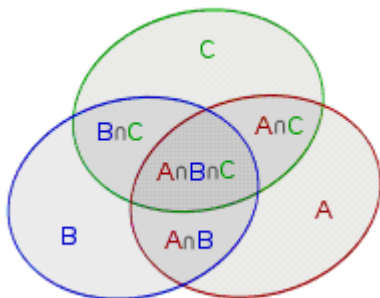
```

分拆数

将 n 分成 k 个部分的分拆, 称为 k 部分拆, 记作 $p(n, k)$

递推式: $p(n, k) = p(n-1, k-1) + p(n-k, k)$

容斥原理



$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |C \cap A| + |A \cap B \cap C|$$

进一步推广:

$$\left| \bigcup_{i=1}^n S_i \right| = \sum_i |S_i| - \sum_{i < j} |S_i \cap S_j| + \sum_{i < j < k} |S_i \cap S_j \cap S_k| - \dots \\ + (-1)^{m-1} \sum_{a_i < a_{i+1}} \left| \bigcap_{i=1}^m S_{a_i} \right| + \dots + (-1)^{n-1} |S_1 \cap \dots \cap S_n|$$

经典例子：不定方程非负整数解

线性代数

矩阵

```

1  template<typename E>
2  struct Matrix{
3      std::vector<std::vector<E> > a;
4      Matrix(int _n): a(_n, std::vector<E> (_n)) {}
5      Matrix(int _n, int _m): a(_n, std::vector<E>(_m)) {}
6      Matrix(int _n, const std::vector<E>& vec): a(_n, std::vector<E>(_n)) {
7          for (int _i = 0; _i < _n; ++_i) {
8              a[_i].assign(vec.begin() + _i * _n, vec.begin() + (_i + 1) * _n);
9          }
10     }
11     Matrix(int _n, const std::initializer_list<E>& vec): a(_n, std::vector<E>(_n)) {
12         for (int _i = 0; _i < _n; ++_i) {
13             a[_i].assign(vec.begin() + _i * _n, vec.begin() + (_i + 1) * _n);
14         }
15     }
16     Matrix(int _n, int _m, const std::vector<E>& vec): a(_n, std::vector<E>(_m)) {
17         for (int _i = 0; _i < _n; ++_i) {
18             a[_i].assign(vec.begin() + _i * _m, vec.begin() + (_i + 1) * _m);
19         }
20     }
21     Matrix(int _n, int _m, const std::initializer_list<E>& vec): a(_n, std::vector<E>(_m)) {
22         for (int _i = 0; _i < _n; ++_i) {
23             a[_i].assign(vec.begin() + _i * _m, vec.begin() + (_i + 1) * _m);
24         }
25     }
26     Matrix(const Matrix<E>& B) {
27         a.resize(B.a.size());
28         for (int _i = 0; _i < a.size(); ++_i) a[_i].assign(B.a[_i].begin(), B.a[_i].end());
29     }
30
31     ~Matrix() {}
32
33     const std::vector<E>& operator [] (int idx) const {return a[idx];}
34     std::vector<E>& operator [] (int idx) {return a[idx];}
35     Matrix<E> operator =(E x) { // 如果不是方阵就全置 0
36         for (int _i = 0; _i < a.size(); ++_i) {
37             a[_i].assign(a[_i].size(), E(0));
38             if (a[_i].size() == a.size()) a[_i][_i] = x;
39         }
40         return (*this);
41     }
42     Matrix<E> operator =(const Matrix<E>& B) {
43         a.resize(B.a.size());
44         for (int _i = 0; _i < a.size(); ++_i) {
45             a[_i].assign(B.a[_i].begin(), B.a[_i].end());
46         }
47         return (*this);
48     }
49     Matrix<E> operator +=(const Matrix<E>& B) {
50         int _n = min(a.size(), B.a.size()), _m = min(a[0].size(), B.a[0].size());
51         for (int _i = 0; _i < _n; ++_i) {
52             for (int _j = 0; _j < _m; ++_j) {
53                 a[_i][_j] += B.a[_i][_j];
54             }
55         }
56     }

```

```

55     }
56     return (*this);
57 }
58 Matrix<E> operator *=(const E& x) {
59     for (int _i = 0; _i < a.size(); ++_i) {
60         for (int _j = 0; _j < a[_i].size(); ++_j) {
61             a[_i][_j] *= x;
62         }
63     }
64     return (*this);
65 }
66 Matrix<E> operator *(const Matrix<E>& B) {
67     if (a[0].size() != B.a.size()) return Matrix<E> (1);
68     Matrix<E> ret(a.size(), B.a[0].size());
69
70     int _n = a.size(), _m = B.a[0].size(), _z = a[0].size();
71     for (int _i = 0; _i < _n; ++_i) {
72         for (int _k = 0; _k < _z; ++_k) {
73             E x = a[_i][_k];
74             for (int _j = 0; _j < _m; ++_j) {
75                 (ret[_i][_j] += x * B[_k][_j]);
76             }
77         }
78     }
79     return ret;
80 }
81 Matrix<E> operator *=(const Matrix<E>& B) {
82     Matrix<E> res((*this) * B);
83     *this = res;
84     return (*this);
85 }
86 Matrix<E> operator +(const Matrix<E>& B) {
87     Matrix<E> ret(*this);
88     return (ret += B);
89 }
90 };
91 template<typename E>
92 Matrix<E> mpow(Matrix<E> n, int k) {
93     Matrix<E> ret(n);
94     ret = E(1);
95     while (k) {
96         if (k & 1) (ret *= n);
97         (n *= n);
98         k >>= 1;
99     } return ret;
100 }

```

线性基

```

1  struct LinearBase{
2      std::vector<int> LBP;
3      LinearBase(): LBP(64) {}
4      LinearBase(int _n): LBP(_n) {}
5
6      void insert(int x) {
7          for (int i = (int)LBP.size() - 1; i >= 0; --i) {
8              if ((x >> i) & 1) {
9                  if (!LBP[i]) {LBP[i] = x; break;}
10                 else x ^= LBP[i];
11             }
12             else continue;
13         }
14     }
15     bool query(int x) {
16         for (int i = (int)LBP.size() - 1; i >= 0; --i) {
17             if ((x >> i) & 1) x ^= LBP[i];
18         }
19         return (x == 0);
20     }
21     int MaxSum() {
22         int ret = 0;

```

```

23     for (int i = (int)LBP.size() - 1; i >= 0; --i) {
24         if ((ret >> i) & 1) continue;
25         else ret ^= LBP[i];
26     }
27     return ret;
28 }
29 };

```

博弈论

Sprague-Grundy Theorem (SG 定理), 将多个游戏的 SG 值异或起来就能表示组合起来的胜负状态. 可以归纳法来证明.

图论

LCA

● 倍增

```

1 void dfs(int u, int fa) {
2     pa[u][0] = fa; dep[u] = dep[fa] + 1;
3     for (int i = 1; i < SP; ++i) pa[u][i] = pa[pa[u][i-1]][i-1];
4     for (int& v: G[u]) {
5         if (v == fa) continue;
6         dfs(v, u);
7     }
8 }
9
10 int lca(int u, int v) {
11     if (dep[u] < dep[v]) swap(u, v);
12     int t = dep[u] - dep[v];
13     for (int i = 0; i < SP; ++i) if (t & (1 << i)) u = pa[u][i];
14     for (int i = SP-1; i >= 0; --i) {
15         int uu = pa[u][i], vv = pa[v][i];
16         if (uu != vv) { u = uu; v = vv; }
17     }
18     return u == v ? u : pa[u][0];
19 }

```

● 树链剖分

```

1 namespace HLD{
2     const int N = 1e5+3;
3     std::vector<std::pair<int, int>> g[N]; // 原本的树
4     int idx[N]; // idx[x] -> 点 x 的 dfs 序号
5     int ridx[N]; // ridx[id] -> dfs 序为 id 的点号
6     int dep[N]; // dep[x] -> 点 x 的深度
7     int siz[N]; // siz[x] -> 以点 x 为根的树的大小
8     int top[N]; // top[x] -> 点 x 所在重链的顶点
9     int fa[N]; // fa[x] -> 点 x 的父亲结点
10    int son[N]; // son[x] -> 点 x 的重儿子
11    int clk = 0; // clk -> dfs 序号
12
13    void init() {
14        for (int i = 0; i <= clk; ++i)
15            g[i].clear();
16        clk = 0;
17    }
18    int lca(int u, int v) {
19        while (top[u] != top[v]) {
20            if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
21            u = fa[top[u]];
22        }
23        if (dep[u] > dep[v]) std::swap(u, v);
24        return u;
25    }
26    void HLDinit(int rt) {
27        std::function<void(int, int)> predfs1 = [&](int u, int f) {
28            siz[u] = 1;
29            int &maxs = son[u] = -1;

```

```

30     for (auto &[to, w]: g[u]) {
31         if (to == f) continue;
32         fa[to] = u, dep[to] = dep[u] + 1;
33         predfs1(to, u);
34         siz[u] += siz[to];
35         if (maxs == -1 || siz[to] > siz[maxs]) maxs = to;
36     }
37 };
38 std::function<void(int, int)> predfs2 = [&](int u, int tp) {
39     top[u] = tp;
40     idx[u] = ++clk, ridx[clk] = u;
41     if (son[u] != -1) {
42         predfs2(son[u], tp);
43     }
44     for (auto &[to, w]: g[u]) {
45         if (to != fa[u] && to != son[u]) predfs2(to, to);
46     }
47 };
48 predfs1(rt, -1);
49 predfs2(rt, rt);
50 }
51
52 void add(int u, int v, int w = 0) {
53     g[u].emplace_back(v, w);
54     g[v].emplace_back(u, w);
55 }
56 }

```

连通性

tarjan

```

1  const int N = 1e5 + 3;
2  std::vector<int> g[N];
3  int dfn[N], dfnx = 0; // dfs 序
4  int low[N]; // 最高点
5  int col[N], colx = 0; // 连通块标号
6  int stk[N], stkt = 0; // 当前链
7  bool instk[N];
8  void tarjan(int u, int f) {
9      dfn[u] = low[u] = ++dfnx;
10     stk[++stkt] = u;
11     for (auto to: g[u]) {
12         // if (to == f) continue;
13         if (dfn[to]) {if (instk[to]) low[u] = std::min(dfn[to], low[u]);}
14         else tarjan(to, u), low[u] = std::min(low[u], low[to]);
15     }
16     if (low[u] == dfn[u]) {
17         ++colx;
18         while (stk[stkt] != u) instk[stk[stkt]] = 0, col[stk[stkt--]] = colx;
19         instk[stk[stkt]] = 0, col[stk[stkt--]] = colx;
20     }
21 }

```

树

虚树

用栈处理形式上与笛卡尔树类似, 按顺序后处理右链部分

```

1  namespace VirtualTree{
2      const int N = 3e5+3;
3      std::vector<std::pair<int, int> > g[N]; // 原本的树
4      int idx[N]; // idx[x] -> 点 x 的 dfs 序号
5      int ridx[N]; // ridx[id] -> dfs 序为 id 的点号
6      int dep[N]; // dep[x] -> 点 x 的深度
7      int siz[N]; // siz[x] -> 以点 x 为根的树的大小
8      int top[N]; // top[x] -> 点 x 所在重链的顶点
9      int fa[N]; // fa[x] -> 点 x 的父亲结点
10     int son[N]; // son[x] -> 点 x 的重儿子

```

```

11  int clk = 0; // clk -> dfs 序号
12
13  void init() {
14      for (int i = 0; i <= clk; ++i)
15          g[i].clear();
16      clk = 0;
17  }
18  int lca(int u, int v) {
19      while (top[u] != top[v]) {
20          if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
21          u = fa[top[u]];
22      }
23      if (dep[u] > dep[v]) std::swap(u, v);
24      return u;
25  }
26  void HLDinit(int rt) {
27      std::function<void(int, int)> predfs1 = [&](int u, int f) {
28          siz[u] = 1;
29          int &maxs = son[u] = -1;
30          for (auto &[to, w]: g[u]) {
31              if (to == f) continue;
32              fa[to] = u, dep[to] = dep[u] + 1;
33              predfs1(to, u);
34              siz[u] += siz[to];
35              if (maxs == -1 || siz[to] > siz[maxs]) maxs = to;
36          }
37      };
38      std::function<void(int, int)> predfs2 = [&](int u, int tp) {
39          top[u] = tp;
40          idx[u] = ++clk, ridx[clk] = u;
41          if (son[u] != -1) {
42              predfs2(son[u], tp);
43          }
44          for (auto &[to, w]: g[u]) {
45              if (to != fa[u] && to != son[u]) predfs2(to, to);
46          }
47      };
48      predfs1(rt, -1);
49      predfs2(rt, rt);
50  }
51
52  int stk[N]; // 栈
53  bool isl[N];
54  std::vector<int> vg[N]; // 虚树
55  std::vector<int> lst; // 当前虚树中的点
56  void buildvt(std::vector<int> pt, int rt = 0) {
57      for (auto x: lst) vg[x].clear(), isl[x] = 0;
58      lst.clear();
59      sort(pt.begin(), pt.end(), [](int u, int v) {return idx[u] < idx[v];});
60
61      int tp = 0;
62      stk[++tp] = rt; lst.push_back(rt);
63      for (auto u: pt) {
64          if (u != rt) {
65              int ll = lca(u, stk[tp]);
66              if (ll != stk[tp]) {
67                  while (idx[ll] < idx[stk[tp-1]])
68                      vg[stk[tp-1]].push_back(stk[tp]), --tp;
69                  vg[ll].push_back(stk[tp]);
70                  if (idx[ll] > idx[stk[tp-1]]) stk[tp] = ll, lst.push_back(ll);
71                  else --tp;
72              }
73              stk[++tp] = u, lst.push_back(u);
74          }
75          isl[u] = 1;
76      }
77      while (tp > 1) {
78          vg[stk[tp-1]].push_back(stk[tp]);
79          --tp;
80      }
81  }

```

```

82
83     void add(int u, int v, int w = 0) {
84         g[u].emplace_back(v, w);
85         g[v].emplace_back(u, w);
86     }
87 }

```

网络流

Dinic

```

1  class Dinic{
2      struct E{
3          int to, cp;
4          E(int t, int c): to(t), cp(c) {}
5      };
6
7      int n, m, s, t;
8      std::vector<E> edges;
9      std::vector<std::vector<int>> > G;
10     int *d, *cur;
11
12     bool BFS() {
13         memset(d, 0, sizeof(int) * (n));
14         std::queue<int> Q;
15         Q.push(s); d[s] = 1;
16         while (!Q.empty()) {
17             int x = Q.front(); Q.pop();
18             for (auto &i: G[x]) {
19                 E &e = edges[i];
20                 if (!d[e.to] && e.cp > 0) {
21                     d[e.to] = d[x] + 1;
22                     Q.push(e.to);
23                 }
24             }
25         }
26         return d[t];
27     }
28
29     int DFS(int u, int cp) {
30         if (u == t || !cp) return cp;
31         int tmp = cp, f;
32         for (int& i = cur[u]; i < G[u].size(); ++i) {
33             E& e = edges[G[u][i]];
34             if (d[u] + 1 == d[e.to]) {
35                 f = DFS(e.to, std::min(cp, e.cp));
36                 e.cp -= f;
37                 edges[G[u][i] ^ 1].cp += f;
38                 cp -= f;
39                 if (!cp) break;
40             }
41         }
42         return tmp - cp;
43     }
44
45     public:
46     Dinic(int nn): n(nn), G(nn), m(0) {d = new int[nn], cur = new int[nn];}
47     ~Dinic() {delete[] d; delete[] cur;}
48
49     void add(int u, int v, int cap) {
50         edges.emplace_back(v, cap);
51         edges.emplace_back(u, 0);
52         G[u].push_back(m++);
53         G[v].push_back(m++);
54     }
55
56     int go(int ss, int tt) {
57         s = ss, t = tt;
58         int flow = 0;
59         while (BFS()) {
60             memset(cur, 0, sizeof(int) * n);

```

```

61         flow += DFS(s, INF);
62     }
63     return flow;
64 }
65 };

```

上下界只需要在建图的时候把上下界之差的出入不平衡的调整一下

最大流 == 最小割

计算几何

二维几何：点与向量

```

1  const double EPS = 1e-9;
2
3  inline int sign(double a) { return a < -EPS ? -1 : a > EPS; }
4  inline int cmp(double a, double b){ return sign(a-b); }
5
6  struct P {
7      double x, y;
8      P() {}
9      P(double _x, double _y) : x(_x), y(_y) {}
10     P operator+(P p) { return {x + p.x, y + p.y}; }
11     P operator-(P p) { return {x - p.x, y - p.y}; }
12     P operator*(double d) { return {x * d, y * d}; }
13     P operator/(double d) { return {x / d, y / d}; }
14
15     bool operator<(P p) const {
16         int c = cmp(x, p.x);
17         if (c) return c == -1;
18         return cmp(y, p.y) == -1;
19     }
20
21     bool operator==(P o) const{
22         return cmp(x,o.x) == 0 && cmp(y,o.y) == 0;
23     }
24
25     double dot(P p) { return x * p.x + y * p.y; }
26     double det(P p) { return x * p.y - y * p.x; }
27
28     double distTo(P p) { return (*this-p).abs(); }
29     double alpha() { return atan2(y, x); }
30     void read() { std::cin >> x >> y; }
31     void write() {std::cout << "(" << x << ", " << y << ")" << std::endl;}
32     double abs() { return sqrt(abs2());}
33     double abs2() { return x * x + y * y; }
34     P rot90() { return P(-y,x);}
35     P unit() { return *this/abs(); }
36     int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); }
37     P rot(double an){ return {x*cos(an)-y*sin(an),x*sin(an) + y*cos(an)}; }
38 };
39
40 struct L{ //ps[0] -> ps[1]
41     P ps[2];
42     P& operator[](int i) { return ps[i]; }
43     P dir() { return ps[1] - ps[0]; }
44     L (P a,P b) {
45         ps[0]=a;
46         ps[1]=b;
47     }
48     bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) > 0; }
49     L push(){ // push eps outward
50         const double eps = 1e-8;
51         P delta = (ps[1] - ps[0]).rot90().unit() * eps;
52         return {ps[0] + delta, ps[1] + delta};
53     }
54 };
55

```

```

56 #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
57 #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

```

字符串

KMP

可以尝试从 *fail* 树的角度理解

```

1  bool kmp(string &s, string &p) {
2      int n = p.size();
3      vector<int> nex(n + 1);
4      for (int i = 1, l = 0; i < n; ++i) {
5          while (l && p[l] != p[i]) {l = nex[l - 1];}
6          if (p[l] == p[i]) nex[i] = l;
7          else nex[i] = ++l;
8      }
9      n = s.size();
10     for (int i = 0, now = 0; i < n; ++i) {
11         while (now && s[i] != p[now]) now = nex[now - 1];
12         if (s[i] == p[now]) ++now;
13         if (now == p.size()) return true;
14     }
15     return false;
16 }

```

manacher

```

1  // 返回处理之后的字符串的极大回文半径，其值减 1 就是该位置的实际极大回文字符串的长度。
2  std::vector<int> Manacher(std::string ss) {
3      std::string s = "#";
4      for (auto ch: ss) s += ch, s += '#';
5
6      int n = s.size();
7      std::vector<int> d1(n);
8      for (int i = 0, l = 0, r = -1; i < n; i++) {
9          int k = (i > r) ? 1 : std::min(d1[l + r - i], r - i + 1);
10         while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
11             k++;
12         }
13         d1[i] = k--;
14         if (i + k > r) {
15             l = i - k;
16             r = i + k;
17         }
18     }
19
20     return d1;
21 }

```

后缀自动机

```

1  struct SuffixAutomaton {
2      static constexpr int ALPHABET_SIZE = 26;
3      int N = 1e5;
4      struct Node {
5          int len;
6          int link;
7          // int next[ALPHABET_SIZE];
8          std::array<int, ALPHABET_SIZE> next;
9          Node() : len(0), link(0), next{0} {}
10     };
11     std::vector<Node> t;
12     int cntNodes;
13     int extend(int p, int c) {
14         if (t[p].next[c]) {
15             int q = t[p].next[c];
16             if (t[q].len == t[p].len + 1)

```

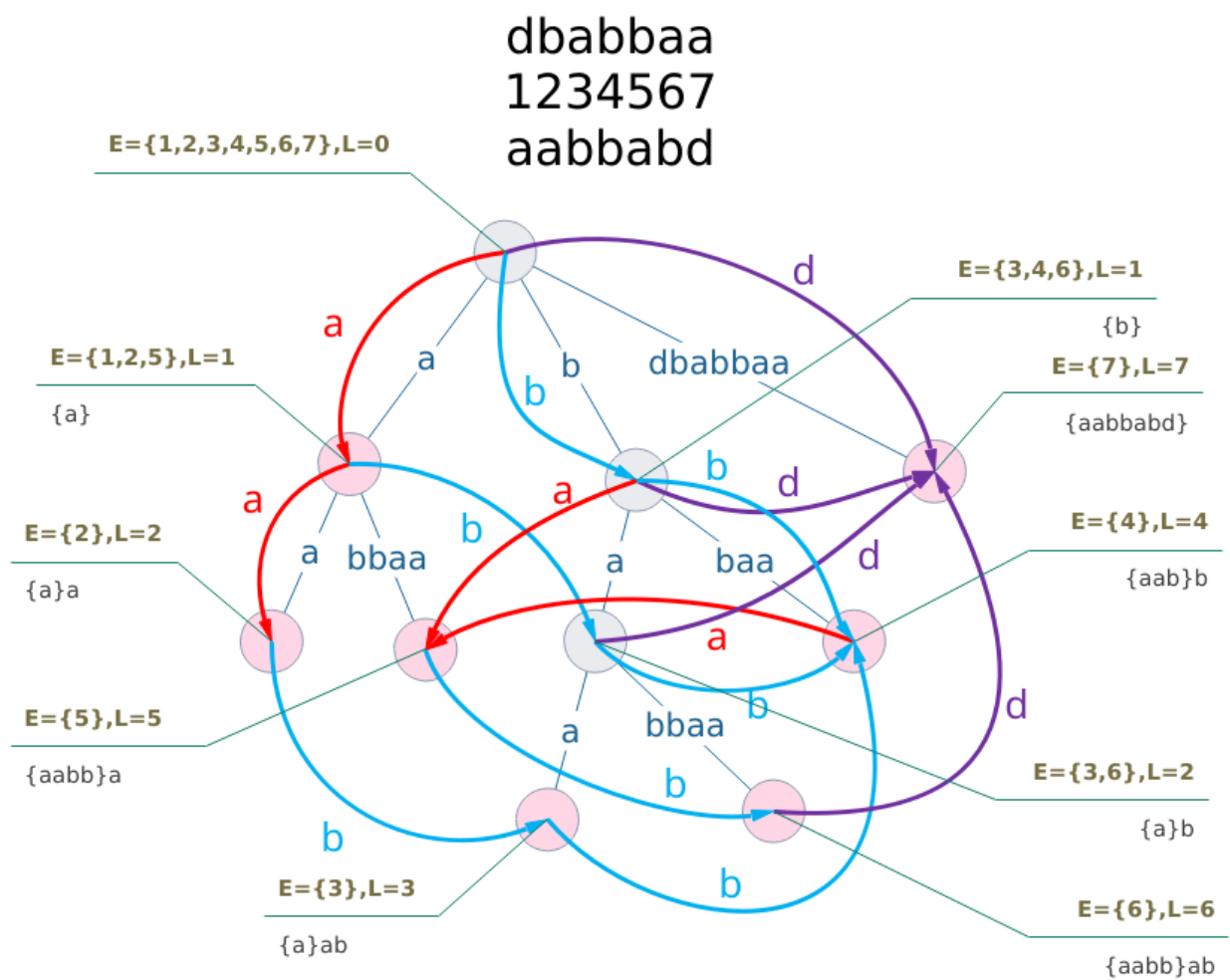



Figure 1: 后缀自动机图解

```

17         return q;
18         int r = ++cntNodes;
19         t[r].len = t[p].len + 1;
20         t[r].link = t[q].link;
21         // std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
22         t[r].next = t[q].next;
23         t[q].link = r;
24         while (t[p].next[c] == q) {
25             t[p].next[c] = r;
26             p = t[p].link;
27         }
28         return r;
29     }
30     int cur = ++cntNodes;
31     t[cur].len = t[p].len + 1;
32     while (!t[p].next[c]) {
33         t[p].next[c] = cur;
34         p = t[p].link;
35     }
36     t[cur].link = extend(p, c);
37     return cur;
38 }
39 SuffixAutomaton(int N_): t(2 * N_), N(N_) {
40     cntNodes = 1;
41     // std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
42     t[0].next.assign(ALPHABET_SIZE, 1);
43     t[0].len = -1;
44 }
45 SuffixAutomaton(string s): t(2 * s.size()), N(s.size()) {
46     cntNodes = 1;
47     t[0].next.assign(ALPHABET_SIZE, 1);
48     t[0].len = -1;
49
50     int p = 1;
51     for (auto ch: s) {
52         p = extend(p, ch-'a');
53     }
54 }
55 };

```

回文自动机

```

1 struct Palindromic_Tree {
2     int nxt[N][30], fail[N], cnt[N];
3     int num[N], len[N], s[N];
4     int last, n, p;
5
6     int newnode(int l) {
7         memset(nxt[p], 0, sizeof(nxt[p]));
8         cnt[p] = num[p] = 0;
9         len[p] = l;
10        return p++;
11    }
12
13    void init() {
14        p = 0;
15        newnode(-1);
16        newnode(0);
17        n = 0;
18        last = 1;
19        s[0] = -1;
20        fail[1] = 0;
21    }
22
23    int get_fail(int x) {
24        while (s[n - len[x] - 1] != s[n])
25            x = fail[x];
26        return x;
27    }
28
29    void add(int c) {

```

```

30     c -= 'a';
31     s[++n] = c;
32     int cur = get_fail(last);
33     if (!nxt[cur][c]) {
34         int now = newnode(len[cur] + 2);
35         if (nxt[get_fail(fail[cur])][c] != 0)
36             fail[now] = nxt[get_fail(fail[cur])][c];
37         else fail[now] = 1;
38         nxt[cur][c] = now;
39         num[now] = num[fail[now]] + 1;
40     }
41     last = nxt[cur][c];
42     cnt[last]++;
43 }
44
45 long long Count() {
46     long long ret = 0;
47     for (int i = p - 1; i >= 0; i--) cnt[fail[i]] += cnt[i];
48     for (int i = 0; i < p; i++)
49         ret = std::max(ret, 1LL * cnt[i] * len[i]);
50     return ret;
51 }
52 } pam;

```

杂项

STL

- copy

```

1  template <class InputIterator, class OutputIterator>
2  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);

```